Implementing an SDN based learning switch to measure and evaluate UDP traffic

Althobyani, M & Wang, X

Original citation & hyperlink:
Althobyani, M & Wang, X 2017, 'Implementing an SDN based learning switch to measure and evaluate UDP traffic' Computers & Electrical Engineering, vol (in press), pp. (in press)
https://dx.doi.org/10.1016/j.compeleceng.2017.07.002

DOI 10.1016/j.compeleceng.2017.07.002
ISSN 0045-7906
ESSN 1879-0755

Publisher: Elsevier

Implementing an SDN based learning switch to measure and evaluate UDP traffic.

MAJED ALTHOBYANI, Faculty of Engineering and Computing, Department of Computing, Coventry University, UK
althobym@uni.coventry.ac.uk, mbth201@gmail.com

Dr. Xingang Wang, Coventry University
Aa9038@coventry.ac.uk

Abstract

The traditional architecture of the network has legacy issues such as manageability, scalability and flexibility. Network administrators do not have much power to control and monitor their networks. Therefore, a new network architecture, so-called software-defined networking (SDN), has emerged to address these challenging issues by redefining the network to make it programmable. SDN relies on decoupling the control plane from the data plane so that it is central and programmable. OpenFlow protocol, which is an open source protocol adopted by the Open Networking Foundation, is empowered to facilitate the communication between controller and switch and to enable agile management of the network. However, OpenFlow-based SDN has two shortcomings that affect the network's performance. First, the communication between the controller and the switch introduces further delay in round-trip time (RTT). Second, the RTT delay results in packet loss or packets arriving at the destination out of order. These two major issues are more likely to occur with UDP traffic due to its characteristics. This paper explains the reasons behind these issues, hypothetically discusses some potential solutions and implements the proposed algorithm. The proposed algorithm reduces the number of packets out of order by increasing the hard timeout time of flow entries when the CPU's usage increases and by gathering statistics from the switch more often. During the demonstration of OpenFlow-based SDN, a POX controller and Mininet are used to emulate the infrastructure needed.

Keywords: software-defined networking, OpenFlow, Mininet, Open vSwitch, POX, packets out of order, RTT.

1. Introduction

Software-defined networking (SDN) revolutionizes the traditional network with the new emerging architecture from a static network to a dynamic network, providing flexibility and scalability to the network. SDN emerged to open network interfaces and to allow the network's devices to be programmable. SDN brings a variety of capabilities that help to address many of the conventional network's issues. A traditional device such as a switch or router has two planes: the data plane and the control plane. The data plane is in charge of forwarding arrived packets based on the forwarding table, such as a routing information base or a forwarding information base. The control plane is part of the device's architecture and is responsible for decision-making and deciding the best route for a packet, based on the knowledge it acquires from the hardware status, dynamic routing protocols or a manual configuration installed by a network administrator. In SDN architecture, the control plane is moved from the packet-switching devices to a logically centralized machine, which is called the controller. Forwarding decisions are made by the controller and then pushed down the rules to the switches in order to perform these rules, whether forwarding, modifying or dropping. This gives the network administrators full control to administer the entire network and manipulate the traffic as required. The network's device is a simple place where flow entries are stored. The OpenFlow protocol, the so-called southbound protocol, facilitates the communication between the control plane and the data plane using a TCP or TLS session, which allows the controller to instruct the network's devices securely. OpenFlow protocol emerged alongside the SDN approach to standardize southbound communication, as adopted by the Open Networking Foundation (ONF) [1]. The data plane has multiple flow tables in which the traffic can be classified to a separate lookup; however, when it comes to physical devices, high memory performance is needed to accelerate the lookup. SDN's architecture has brought many benefits to the network, such as a programmable interface that gives network administrators the ability to develop their own rules, as well as centralized control to apply those rules in an agile manner. Despite this, there are some drawbacks associated with SDN's approach based on OpenFlow. The crucial one is the fact that the network's performance is affected by the performance of the switch, the controller or OpenFlow protocol. The performance is affected each time the controller instructs the switch on how to deal with a new arrival packet. Therefore, UDP traffic experiences packet loss or packets out of order at the destination due to the number of packets waiting at the controller to be routed.

The remainder of this paper is instructed as follows: Section 2 provides the literature review on SDN. Section 3 describes the technical background of UDP traffic in OpenFlow network. Section 4 collects and analyses UDP traffic by experiment. The design and the implementation is detailed in section 5, followed by Section 6, where the analysis and results are given. Finally, Section 7 concludes this work.

2. Literature Review

Networks are complex and difficult to manage because many kinds of appliances are involved, such as switches, routers, firewalls and servers. Therefore, the design and management of the network have become more innovative with the assistance of SDN. Over the last few years, the adoption of virtualization networks and the growing concentration on software-defined data centres (SDDCs) have resulted in a drive towards relying on SDN functionality. One of the key characteristics of SDDC is the virtualization of the data centre's infrastructure and delivering as a service. Another key characteristic is automation of the control of data centre applications and services through a management system [2]. Consequently, SDN is used to address legacy network challenges through a data centre based on the Open Data Center Alliance, which provides the concise characteristics of modern network requirements as follows [3]:

- Adaptability: networks have to adapt and respond to application requirements, business needs and network policies dynamically.
- Automation: policy changes have to be automated and automatically propagated to the entire network, thus minimizing manual work and mistakes.
- Maintainability: each new feature, such as software updates and patches, must be seamlessly introduced with a minimum of operational disruption.
- Model management: software management has to permit the management of the network at the model level, rather than practically implementing changes by reconfiguring each network node.
- Mobility: control functionality must feature mobility, including remote users, mobile devices and virtual servers.
- Integrated security: network applications must seamlessly integrate security as a main service, rather than an additional solution.
- On-demand scaling: implementations must be able to scale the network, and their services have to be available for on-demand requests.

2.1. SDN Architecture

SDN opens network interfaces that enable the software to control the connectivity by decoupling control from traffic forwarding, logically centralizing the network and abstracting the underlying network infrastructure from applications [4]. As a result, enterprises and providers obtain first-time programmability, automation and control of the network and can therefore build a flexible and scalable network that adapts to changing business needs.

Figure 1 illustrates the elements of SDN architecture. The elements can be represented as components of different layers [5]. Each layer has its own specific role. The intelligence of the network is logically centralized in an SDN controller, which maintains the entire network. Consequently, it appears as a single and logical switch to the application or management layer. Therefore, this simplifies the network design and operation, as well as the network devices themselves. Thus, they no longer need to understand and process the complexity of protocols;

instead, the SDN controller dictates instructions to the network's devices, and they just accept them.

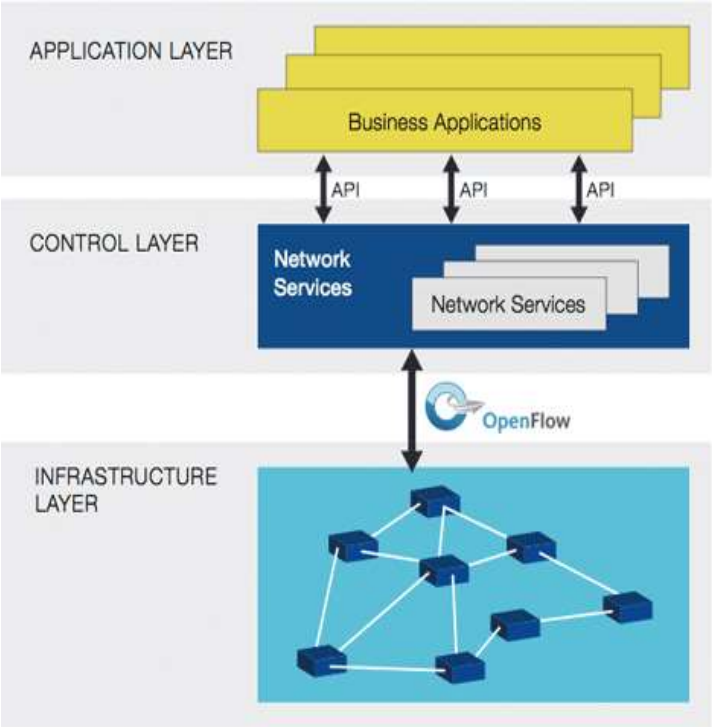


Figure 1: SDN architecture [6]

3. Background

Based on the latest OpenFlow specification (1.3) [7], an OpenFlow switch comprises one or more flow tables and a group table, which perform lookups and forward packets, and one or more OpenFlow channels, which are connection interfaces running across TCP or TLS sessions to communicate with an external controller, as shown in Figure 2.

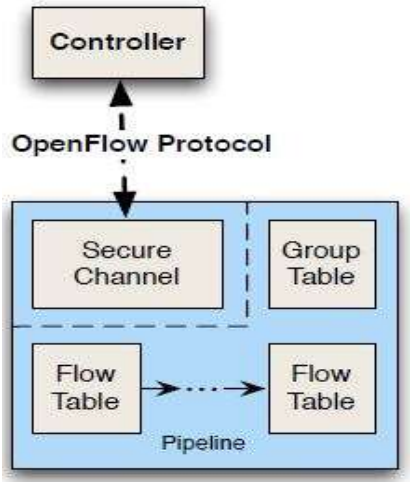


Figure 2: OpenFlow switch components [7]

Using OpenFlow protocol, flow entries can be added, modified and deleted reactively, whereby the controller dynamically learns where devices are in the topology and then pushes the flow entries after receiving a packet from a switch. This can also be done proactively, whereby flow entries are built before traffic arrives; for example, a switch is reprogrammed to drop a packet from computer1, so once computer1 sends a packet, the switch will drop it without the controller being involved. Moreover, the controller can proactively write flow entries to the switches in the network in order to build a flow path for specific devices. In the proactive environment, the controller can be away from the switches, helping to reduce latency in the network and packet loss in UDP traffic. A hybrid mode exists that combines the two, whereby the controller proactively pushes all known flow entries into the switches and reactive mode is on. Therefore, this mode gives the controller the ability to deal with mismatched packets. Matching begins from the first flow table and might continue to the additional table of the pipeline, according to the configuration and based on priority order. If a matching flow entry is found in a flow table, then the actions linked to the flow are executed. If not, the action is dependent on the table-miss flow entry configuration; for example, the packet might be dropped or forwarded to the controller. Packet match fields are extracted from the packet and used for the table lookups, depending on header field information. The resulting delay in round-trip time (RTT) is caused by two situations that miss-table flow entry and the active mode controller, where all packets are sent to the controller for decision-making. There are three assumptions related to the diverted packet route [8]:

1- The packets arrive at a rate equal to or lower than the rate at which the controller switches each packet.
2- The hard timeout time for each flow table insertion is equal to or greater than the flow's duration.
3- The idle timeout time for each flow entry is equal to or greater than the rate at which packets of the flow arrive.

The first assumption is less likely to occur in enterprise networks based on current data transfer speeds. Consequently, more than one packet will have a route to the controller for decision-making, which increases the delay due to RTT. The increase of the delay depends on the performance of the controller and the queue of packets waiting to be routed by the controller. Therefore, the time of arrived packets decreases, and the delay goes up.

UDP traffic is more likely to suffer from packet loss or packets being out of order at the destination [9]. This is because there is no handshake before the packets come and the flow is established, which is unlikely with TCP traffic. UDP traffic does not pause or wait: the traffic bursts immediately, and the flow may not be established at that point. In UDP measurement [10], when a burst of traffic is pushed to the network, switches will not be ready in time, so, if all packets are switched, there will be packet loss as a result. Thus, a model has been introduced that can be used to estimate the packet sojourn time and the probability of lost packets [10].

4. Experiment

This section experiments and measures OpenFlow-based SDN in terms of RTT delay and UDP packets out of order, comparing an OpenFlow network and a non-OpenFlow network.

4.1. Test Method

In order to test OpenFlow's performance in terms of UDP traffic, RTT and the number of packets out of order will be measured as detailed below [11].

4.1.1. RTT

The communication between the controller and the switch introduces an additional delay in RTT for the first packet into the controller. Thus, this additional delay affects the performance. The ping transaction is used to measure the delay when there are no flow entries in the switch. A ten-second ICMP request is sent from Host 1 to Host 2 with a one-second interval, and the hard timeout time is set to one second to make sure the installed flow entry will be removed after one second.

4.1.2. UDP Traffic

As UDP is a connectionless protocol, it is more likely to lose packets or have packets arrive out of order at their destination through transmission. The goal is to study if the delay caused by the communication between the controller and the switch affects the traffic. The iPerf tool is used to generate UDP traffic and measure the number of packets out of order [12].

4.2. Testbed

A topology has been created as shown in Figure 3, which consists of two hosts and a controller, as summarized in Table 1. The topology works on two virtual machines created by VirtualBox software. One of them is running Mininet, which is an emulator used to create a real virtual network, running the real kernel of the switch and the application code on a single virtual machine [13]. The other is running a POX controller, which is an SDN controller based on the Python language [14]. Open vSwitch is software in the Linux kernel used to create virtual switches, with support for OpenFlow protocol [15]. The purpose of using a dedicated machine for the controller is to prevent Mininet CPU utilization from affecting the controller's performance.

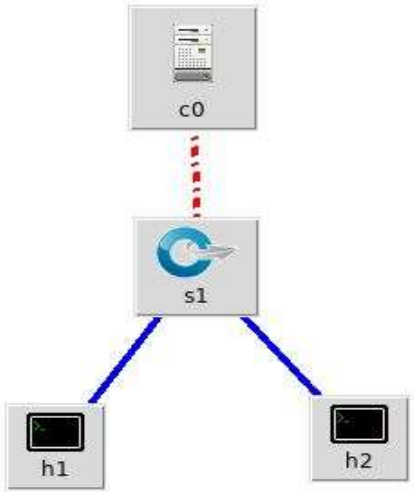


Figure 3: Experiment topology

Device	Host	Virtual Machine
OS	Windows 10 Pro 64-bit	Ubuntu 14.04 64-bit
CPU	Intel Core i7-3667 2.00GHZ	1 core
RAM	8 GB	1 GB
Virtualization	VirtualBox 5.0	KVM

Table 1: Host's and virtual machines' specifications

4.3. Measurement and Analysis

A. RTT delay: the communication between the control plane and the data plane is affected by the RTT delay. Therefore, a ping matrix has been used to measure the time in milliseconds in reactive mode, proactive mode and non-OpenFlow. A ten-second ICMP request was sent from Host 1 to Host 2 with a one-second interval, and the hard timeout time was set to one second.

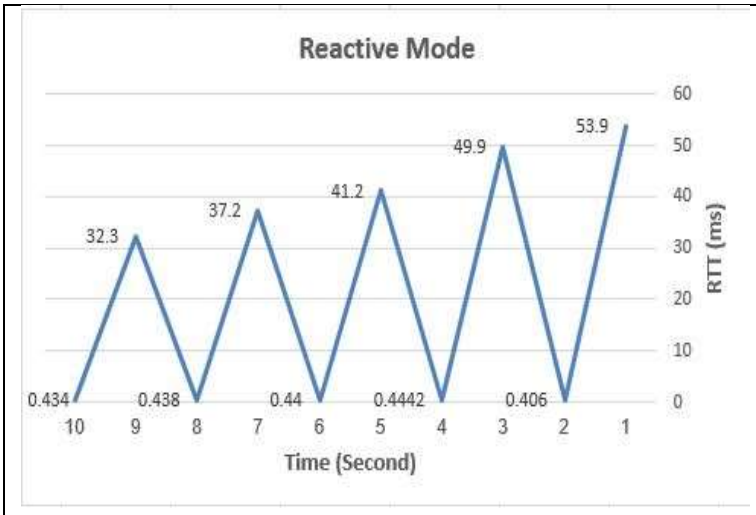


Figure 4.1: Reactive mode

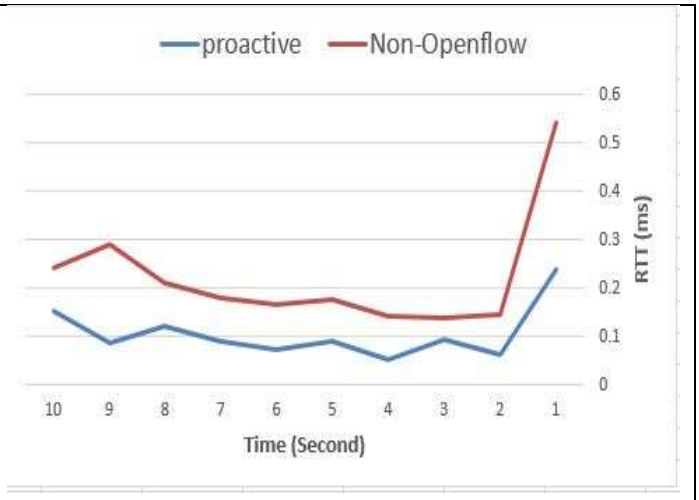


Figure 4.2: Proactive mode and non-OpenFlow

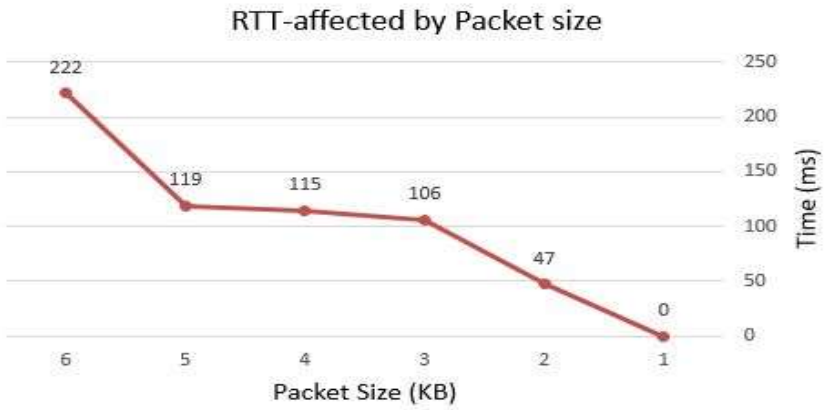


Figure 4.3: RTT increases as packet size increases in the OpenFlow network

As a result, Figure 4.1 shows that each time a packet has to reactively travel to the controller for decision-making, due to the hard timeout time being set to one second, it takes longer than a packet that matches the flow entry. This delay results from exchanging messages between the controller and the switch in order to insert a flow for the packet. This is opposed

to proactive mode, where a path is set before a packet is sent. Therefore, the controller is no longer involved, resulting in no delay in RTT, as well as in the non-OpenFlow network, as depicted in Figure 4.2. Consequently, this is an indication that the OpenFlow messages exchanged between the data and control planes introduce the delay. Figure 4.3 shows that the increase of packet size resulted in growing RTT delay.

B. Packets out of order: UDP traffic was generated and travelled from Host 1 to Host 2, sending 1,460 bytes at a predetermined bit rate for 30 seconds. The following command was executed to do so:

iPerf commands:	
iPerf parameters: s: refers to server u: refers to UDP traffic I: refers to the interval time in seconds p: refers to the port use b: refers to the bandwidth t: refers to the duration in seconds e: refers to the number of packets per second	
Command for preparing Host 1 as a server	Iperf –s –u –I 1 –p 5001
Command for preparing Host 2 as a client	Iperf –c 10.0.0.1 –u –b 1m –p 5001 –I 1 –t 30 -e

The following figures illustrate the results, as summarized in Table 2:

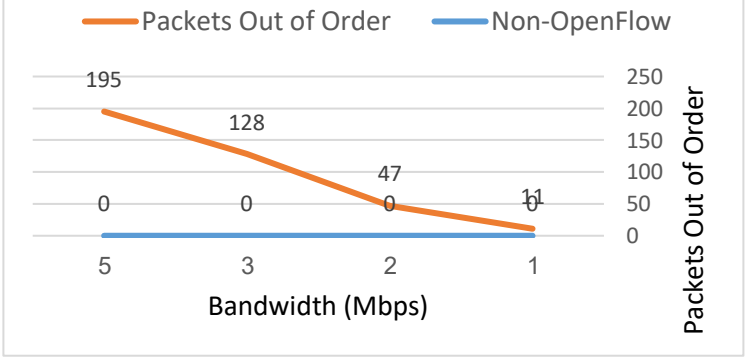


Figure 5.1: Packets out of order in OpenFlow and non-OpenFlow networks

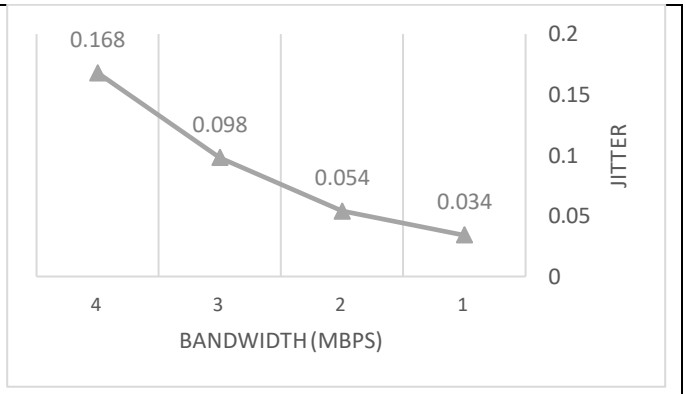


Figure 5.2: Jitter increases as bandwidth increases

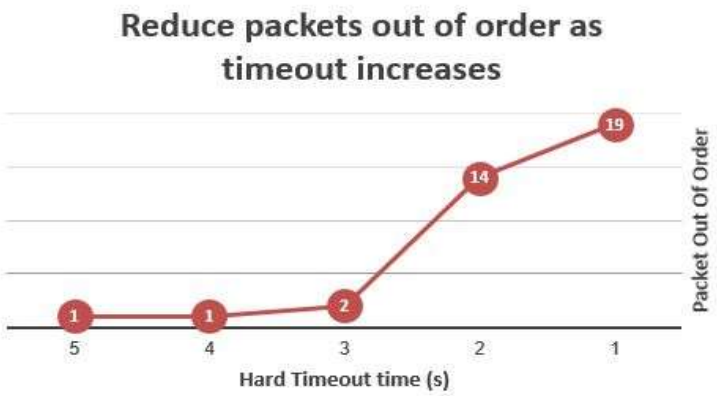


Figure 5.3: Reduced packets out of order as hard timeout time increases

Bandwidth (Mbps)	Packets Out of Order	Jitter (ms)	Packets Sent	Packets per Second
1	11	0.034	2552	85
2	47	0.054	5104	170
3	128	0.098	7654	255
5	195	0.168	12757	425

Table 2: Relation of packets out of order to the packet rate

Figure 5.1 shows that there is a connection between the number of packets out of order and the packet rate. Thus, increasing the packet rate increases the number of packets out of order, as the UDP traffic flows to the controller before installing a flow entry or path for traffic. Moreover, when the sender bursts packets at a high bandwidth, the switch will buffer all packets until the flow table gets full and forwards all packets to the controller for decision-making. Therefore, there will be more packets waiting at the controller to be routed, where packet loss or packets out of order occurs. As a comparison with non-OpenFlow, there were no packets that reached their destination out of order, as the controller was no longer involved. The jitter is relational to the packet rate at the switch, as shown in Figure 5.2. Figure 5.3 shows that if the hard timeout time is increased, then the number of packets out of order will be reduced, as the flow entry will stay longer at the switch, matching packets without the controller intervening.

In summary, compared with the non-OpenFlow network, the performance of the OpenFlow network is affected by the RTT delay.

5. Design of Implementation

This section outlines some potential solutions to the problem and describes the proposed algorithm.

5.1. Hypothetical Solutions

This subsection discusses some solutions that might help to reduce the number of packets out of order in UDP traffic and the disadvantages of them, as follows:

A. Increasing the hard timeout time: this means that flow entries will stay longer at the switch and that most of the packets will be routed directly to their destinations without the controller. The drawback of this solution is that the flow entry of the switch has a limited number of entries. Thus, when it gets full, the controller will be forced to gather statistics to become aware of the rules that have to be dropped, which causes more delay, resulting in increasing packet loss.

B. Deployment of a hybrid mode: the advantage of this approach is that the controller will proactively install the rules into the switch and simultaneously be reactive in case of a packet that does not match the rules. Thus, the controller will not affect the network performance. The disadvantage of this solution is that the network administrator should be aware of all traffic routes in the network.

C. Using idle timeout time instead of hard timeout time: idle timeout time could be very useful, as the flow entry would not be removed from the flow table while it is being matched. In Figure 6, the hard timeout time was set to zero, meaning no hard timeout time was used, and the idle timeout time was set to one second. The UDP traffic was sent from Host 1 to Host 2 at a bandwidth of 5Mbps for 30 seconds.

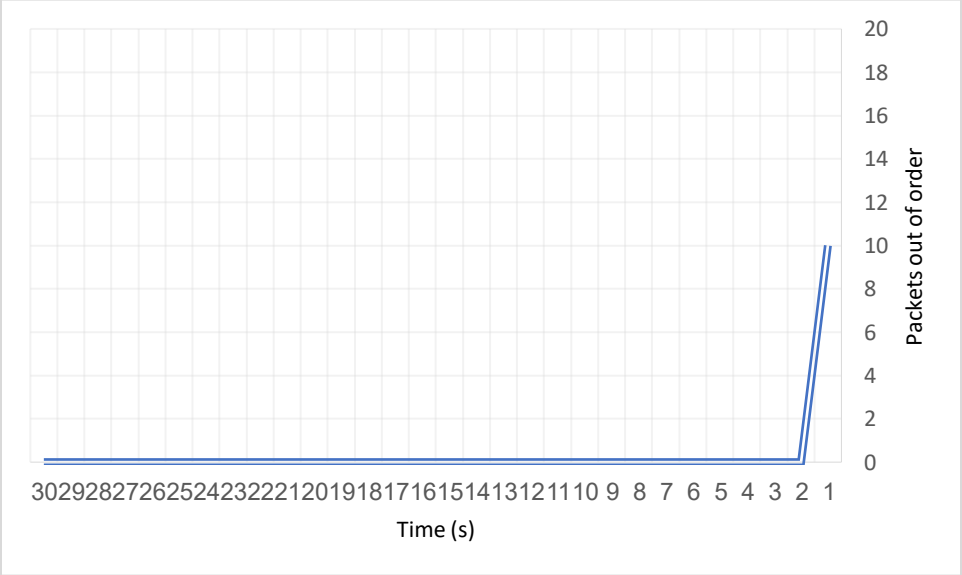


Figure 6: Decreased number of packets out of order through using just idle timeout time

As seen in Figure 6, packets out of order occurred at the beginning of the stream, but after the rule had been installed and after matching, there were no packets out of order. The drawback is that the network might be vulnerable to malicious flow entry because if the inter-arrival time of flows is less or equal to the idle timeout time, the flow will never expire, which increases traffic without the controller being informed.

5.2. The Proposed Algorithm

The algorithm [3] helps to increase the performance of specific flows that are provided to the controller by the network administrator. This allows the controller to be aware of the flows that need special handling or high priority. In order to achieve effective performance, the algorithm will execute the following two operations:

- The controller's CPU consumption will be monitored at a frequency equal to the smallest timeout; thus, if the CPU utilization increases, the hard timeout time of specific flows will be increased. The controller will gather statistics from the switch based on the availability of flow table spaces. Therefore, if the flow table is going to get full, the controller will gather statistics more often to avoid it getting full.
- The controller will be aware of the topology that is created by the network administrator: its rules will be installed and route formation will be performed based on finding the best paths to the destinations.

5.3. Testing Scenarios of the Algorithm

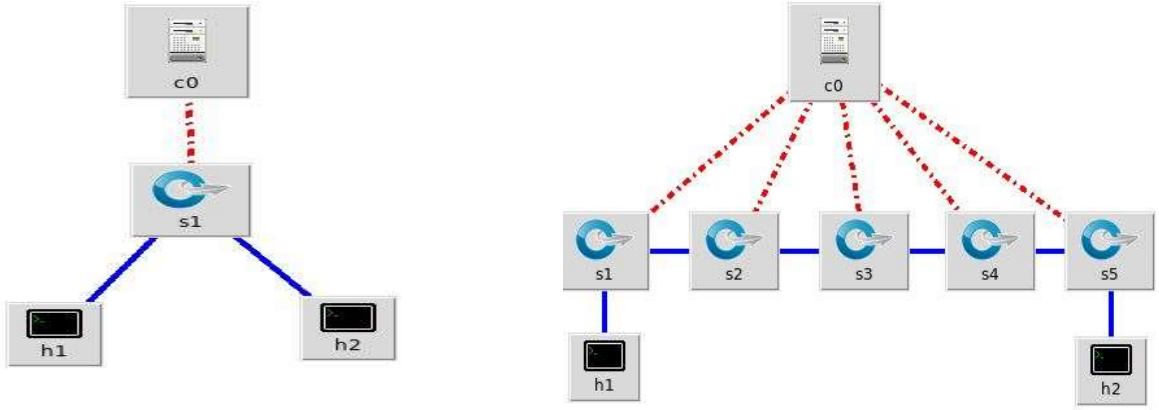
The scenarios' summary and topologies are shown in Figure 7, and the details are listed as follows:

- Scenario 1: the L2_learning switch application was running on the POX controller with no algorithm used. Host 2 sent 5Mbps UDP traffic to Host 1 for 30 seconds, and the hard timeout time was set to one second. The controller's CPU load was increased by 10% every ten seconds. The aim of this experiment was to compare it with Scenario 2.
- Scenario 2: the purpose of this scenario was to examine the effect of increasing the hard timeout time when the controller's CPU was overloaded. Therefore, when the CPU load exceeded 75%, the hard timeout time was raised to two seconds.
- Scenario 3: the topology consisted of five switches connected sequentially and two hosts: Host 1 connected to Switch 1, and Host 2 connected to Switch 5. 5Mbps UDP traffic travelled from Host 1 to Host 2 for 30 seconds, and the hard timeout time was set to one second. The controller's CPU load recorded an increase every ten seconds. The point of the experiment was to compare it with Scenario 4.
- Scenario 4: this scenario examined the performance of mixing hard timeout time and idle timeout time. When a packet was sent to the controller for decision-making, the

controller sent the flow_mod packet to all switches, with a one-second hard timeout time for the first switch that sent the packet in and a one-second idle timeout time for the remaining switches that the packet had to go through in order to reach its destination.

Scenarios	1 and 2	3 and 4
Number of Hosts	2	2
Number of Switches	1	5
Controller	POX	POX
Traffic	5Mbps	5Mpbs
Packet Size	1,470 B	1,470 B
Time	30 sec	30 sec

Table 3: Summary of scenarios



(a) Scenarios 1 and 2 (b) Scenarios 3 and 4

Figure 7: Scenario topologies

6. Implementing and Analysing the Algorithm Results

This section presents the emulation of the algorithm's implementation that was performed according to the testing scenarios described in the previous section. It also analyses the results of the algorithm's implementation.

6.1. Emulation of the Implementation

Based on the testing scenarios, described in Section 4, the POX controller was run in an independent virtual machine and the learning switch application, L2_learning.py, was executed to perform packet switching. The hard timeout and the idle timeout were adjusted in L2_learning.py to meet the scenarios' needs. Mininet was run in another virtual machine, which

gave the ability to emulate the topologies required for the scenarios. For Scenarios 1 and 2, a simple command was executed in Mininet, emulating a single switch with two hosts connected. For Scenarios 3 and 4, a custom topology, 5s2h.py, was created to meet their needs. UDP traffic was generated by iPerf and repeated using two streams running simultaneously in order to observe the controller under more stress.

6.2. Results and Analysis

In Scenario 1, the hard timeout time was set to one second; thus, most of the packets were forwarded to the controller. This resulted in the controller's CPU being overloaded, as shown in Figure 8.1, and taking longer to come to its decisions. Therefore, the number of packets out of order increased, as seen in Figure 8.2, and the high increase in the last ten seconds was related to CPU overload. On the other hand, in Scenario 2, when the controller's CPU was overloaded, the hard timeout time was increased to two seconds. Therefore, flow entries stayed longer at the switch, which reduced the number of packets travelling to the controller, and the CPU usage gradually decreased. Consequently, this slightly decreased the number of packets out of order, as seen in Figure 8.2.

In Scenarios 3 and 4, the mixing of the hard timeout time and the idle timeout time presented enhanced performance in terms of the number of packets out of order, as shown in Figure 8.3. This result was expected, as increasing the hard timeout time or mixing it with the idle timeout time allows an OpenFlow network to work with no controller involved more often. The proposed algorithm helps to mitigate the impact of increasing the hard timeout time per flow by monitoring and managing the flow table in the switch. Moreover, if a flow rule entry expires in the first switch, using hard timeout, it can be removed from the rest of the switches due to inactivity.

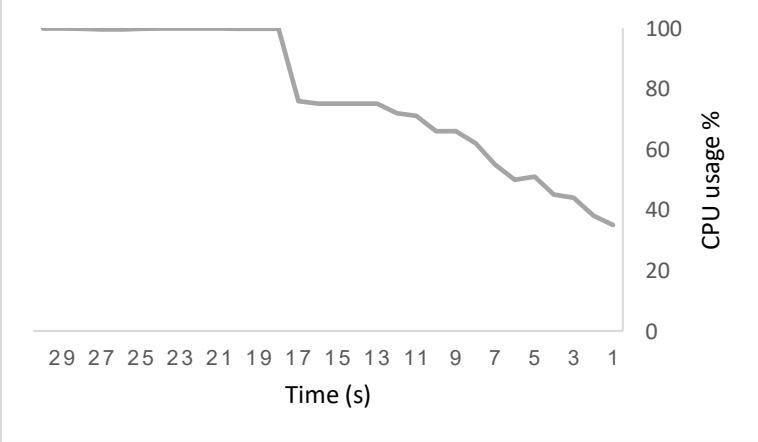


Figure 8.1: Scenario 1 CPU usage

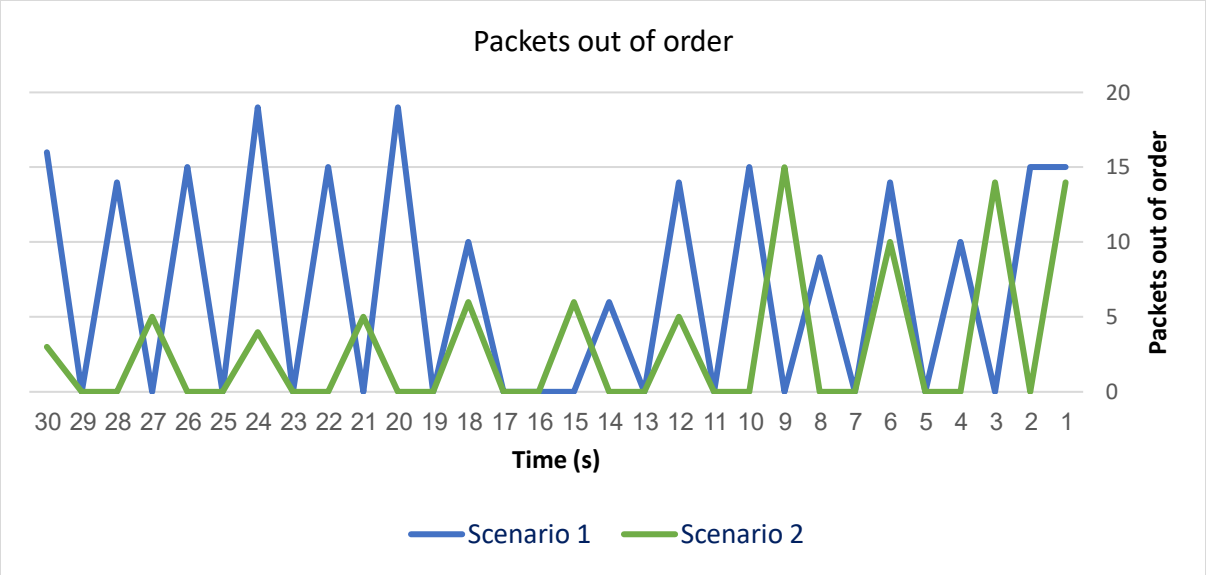


Figure 8.2: Packets out of order in Scenarios 1 and 2

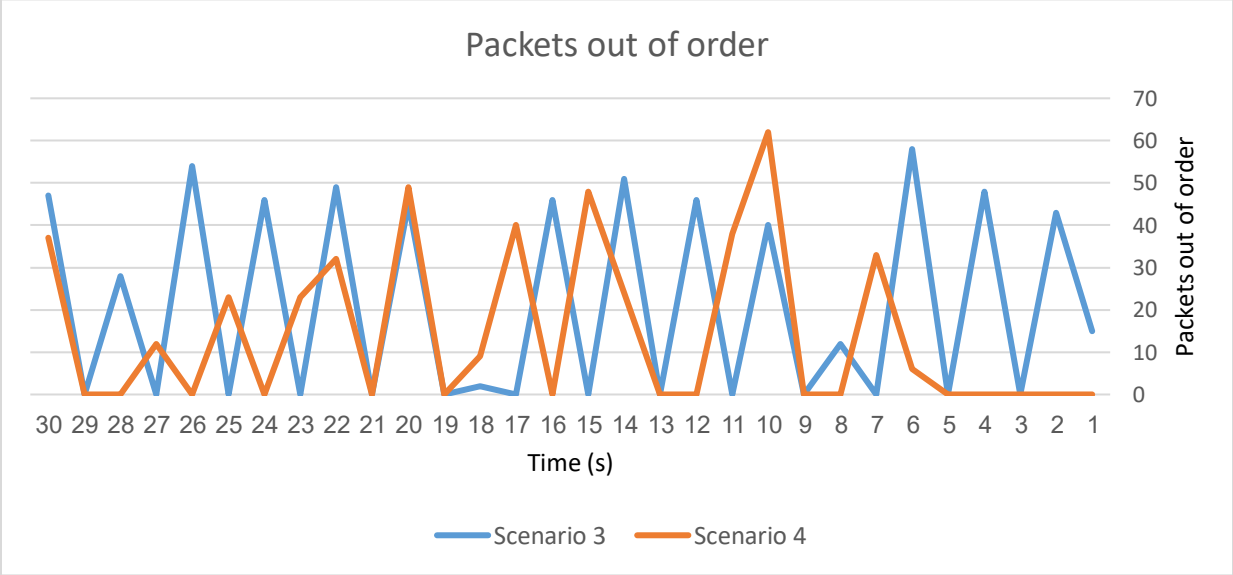


Figure 8.3: Scenarios 3 and 4

7. Conclusion

Packets out of order are an ongoing issue associated with UDP traffic, as UDP packets do not reorder themselves at their destinations. Moreover, in an OpenFlow network, the RTT delay increases the number of packets out of order and the amount of packet loss due to the fact that the controller does not queue packets until it has set paths for them. Consequently, when packets are bursting to a switch, the switch will ask the controller for their destinations to pass them. Thus, the messages exchanged between the switch and the controller introduce a delay, and the sender continues sending packets. In this situation, the switch buffer will get full of flow table entries and will direct all the packets to the controller, where most packet loss occurs, or will forward them out of order. In order to avoid these issues, the timeout time (whether hard or idle) has a big role to play in reducing the delay in RTT if it is managed efficiently, as proven in this paper by using the algorithm. The algorithm relies on amending the timeout time when it is needed. For example, when the hard timeout time increases, the controller must monitor the flow table in the switch in order to avoid it getting full. Thus, it showed an enhancement of the network performance. However, the algorithm is not scalable when it comes to a mesh topology because the traffic has more paths to be forwarded through. This paper has concluded that the RTT delay affects the performance each time the switch has to forward packets to the controller for decision-making.

Acknowledgments

This work has been supervised by Dr Xangang Wang from Coventry University, UK.

References

1. Open Networking Foundation (2016). *Home – Open Networking Foundation*. [online] Available at: https://www.opennetworking.org/ [Accessed 3 Sep. 2016].
2. Citrix (2014). *An Introduction to Software-Defined Networking*. Citrix.
3. Open Data Center Alliance (2014). *Open Data Center Alliance: Software-Defined Networking Rev. 2.0*. Open Data Center Alliance, Tech. Rep.
4. Kreutz, D., Ramos, F., Esteves Verissimo, P., Esteve Rothenberg, C., Azodolmolky, S. and Uhlig, S. (2015). Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 103(1), pp.14-76.
5. Open Networking Foundation (2016). *SDN Architecture.* 1st ed. [eBook] Available at: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-521_SDN_Architecture_issue_1.1.pdf [Accessed 11 Apr. 2016].
6. Open Networking Foundation (2016). *Software-Defined Networking (SDN) Definition – Open Networking Foundation*. [online] Available at: https://www.opennetworking.org/sdn-resources/sdn-definition [Accessed 8 May 2016].

7. OpenFlow Specification (2012). *OpenFlow Switch Specification*. 3rd ed. [eBook] Available at: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf [Accessed 2 Jun. 2016].
8. Isaia, P. and Guan, L. (2015). *OpenFlow Performance Enhancement Algorithm Using Dynamic Installation and Management*. Loughborough University.
9. Jarschel, M., Oechsner, S., Schlosser, D., Pries, R., Goll, S. and Tran-Gia, P. (2011). Modeling and Performance Evaluation of an OpenFlow Architecture. *Proceedings of the 23rd International Teletraffic Congress*, [online] pp.1-7. Available at: http://dl.acm.org/citation.cfm?id=2043470 [Accessed 15 Jun. 2016].
10. Curtis, A., Mogul, J., Tourrilhes, J., Yalagandula, P., Sharma, P. and Banerjee, S. (2011). DevoFlow. *ACM SIGCOMM Computer Communication Review*, 41(4), p.254.
11. Open Networking Foundation (2014). *Migration Tools and Metrics*. 1st ed. [eBook] Available at: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/migration-tools-and-metrics.pdf [Accessed 18 Jun. 2016].
12. Mininet (2016). *Mininet: An Instant Virtual Network on your Laptop (or Other PC) – Mininet*. [online] Available at: http://mininet.org/ [Accessed 30 Jun. 2016].
13. OvS (2014). *Open vSwitch*. [online] Available at: http://openvswitch.org/ [Accessed 30 Jun. 2016].
14. iPerf (2016). *iPerf – The TCP, UDP and SCTP Network Bandwidth Measurement Tool*. [online] Available at: http://iperf.fr/ [Accessed 29 Jun. 2016].
15. POX (2016). *POX Wiki.* [online] Available at: https://openflow.stanford.edu/display/ONL/POX+Wiki [Accessed 30 Jun. 2016].

Vitae

MAJED ALTHOBYANI achieved a Master's degree in Network Computing from Coventry University, UK, in 2016. He received a Bachelor's degree in Computer Technology in 2012 from the College of Technology, Riyadh. He is interested in virtualization and cloud computing. His current research is in SDN, and he has recently deployed NSX from VMware, which is considered an SDDC.

Dr. Wang is currently a senior lecturer in the School of Computing, Electronics, and Math at Coventry University, UK. He worked in Centre for Security, Communications and Network Research, Univeristy of Plymouth for 5 years as a lecturer. He holds the Microsoft Certified System Engineer professional qualification, is a CCNA training instructor and is a member of IEEE and IET.